1P−7　密結合マルチプロセッサ実験システム MUSTARD-POT のトレース方式

広屋 修一
日本電気（株）

1. はじめに

我々は密結合マルチプロセッサ上で動作するリアルタイムUNIX **MUSTARD** [1] を開発しているが、このOS及びアプリケーション・プログラムのデバッグ/性能評価を行うために、 32bitマイクロプロセッサV70を8台まで搭載可能な密結合マルチプロセッサ実験システム **MUSTARD-POT** を開発した。本報告では、MUSTARD-POTの特徴である大容量ハードウェア・トレーサと、このトレース内容から実行された命令系列を推定するアルゴリズムについて述べ、その後、マルチプロセッサ用のトレース方式について述べる。

2. トレーサ・ボードのハード構成

マルチプロセッサ環境でのプログラムの挙動は複雑であり、動的な実行履歴を得ることが、デバッグ及び評価に有効である。この種のトレースは、 Logic Analyzer や In-Circuit Emulator によっても行えるが、マルチプロセッサ環境では、これらの計測機の個数がプロセッサ台数分必要となり、費用がかかるとともに操作が繁雑になる。そこで、我々は MUSTARD-POT (図1) の各プロセッサごとに、 64K バス・サイクルのトレース情報が格納可能な大容量トレーサを付加して、これらをデバッグ・モニタから制御することにした。このトレーサ・ボードは不要な時に、プロセッサ・ボードから取り外すことが可能である。

トレース情報は、全体で88bitで、内容はV70 の入出力信号の address(32bit), data(32bit), status(9bit), interrupt(3bit) と V70 の動作クロックでカウントアップされる時間計測用 counter(8bit) とトレーサ・ボードのパネル面から入力可能な option(4bit) 信号からなる。

トレース・トリガはソフトによる実行開始/停止の要求及びトレース・バッファ・フル事象のみとし、複雑なイベント指定はブレーク・ポイントとの組み合せで実現することにした。 V60用のOSデバッグ・システムを開発した経験から、複雑なトレース機能はめったに使われず、むしろ大容量のトレーサへの要求が大きかったことを考慮して設計した。トレースの実行開始/停止の要求は、自プロセッサからだけでなく、他のプロセッサから1つのトレーサまたはすべてのトレーサを一斉に制御できるようにしている。

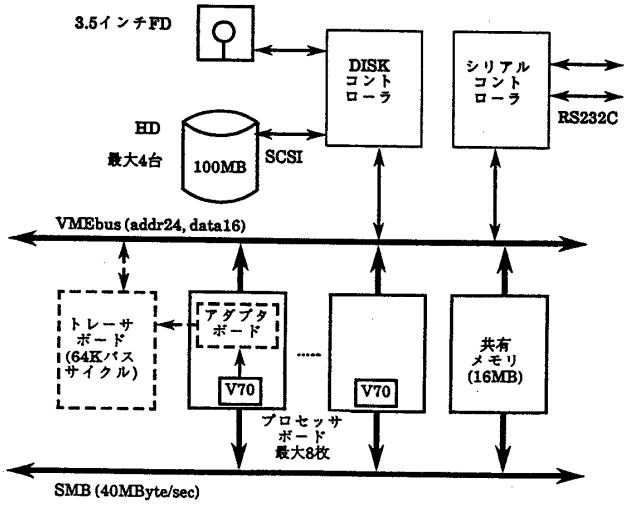

図1. MUSTARD-POT の全体構成

3. 命令レベルへの再構成

各プロセッサに搭載されているトレーサは、単にバス・サイクルごとのアドレス/データなどを記録しているだけなので、性能評価などではそのまま利用できるが、デバッグなどに利用するのはむずかしい。デバッグに利用するためには、これらのデータから命令レベルの動作系列に再構成したいが、 CPUの内部の詳細なトレース結果がないため、プリフェッチした命令を実際に実行したかどうかを正確に判定することはむずかしい。

我々はこの問題に対して、バス・サイクル履歴から実際に実行した命令だけのシーケンスを推定するアルゴリズムを開発した。限られた情報から行うため完全に推定することはできないが、実用上は問題ないことを実用アプリケーション・プログラムのトレース・データを用いて確認している。

3.1 再構成のための仮定

次節で述べる再構成アルゴリズムは、プロセッサの挙動に仮定をおいている。今回は、V70用に実装したが、以下のような仮定を満たす（それほど特殊ではない）アーキテクチャであれば、同様のアルゴリズムを適用可能である。

(1) 分岐後の最初のコード・フェッチを識別可能

(2) 割込み発生時のPCのPUSHのサイクルが推定可能

3.2 再構成のアルゴリズム

以下に再構成のアルゴリズムについて示す。

Trace Mechanism for Multiprocessor System MUSTARD-POT
Shuichi HIROYA
NEC Corporation

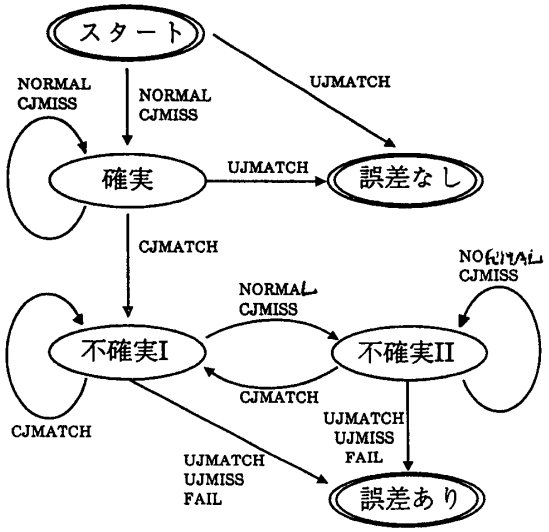

図2. 通常ブロックの解析アルゴリズム

(1) 分岐から分岐、分岐から割込みまでに存在するコードを先読みする。それぞれのコード群を通常ブロック、割込みブロックと呼ぶ。また、同時に次のコード・ブロックの先頭バス・サイクルに存在するアドレスを先読みする。このアドレスを次ブロック先頭アドレスと呼ぶ。

(2) 割込みブロックの場合、頭から逆アセンブルしていき、割込み発生時にPUSHしたPCの値を越えないアドレス部分までが実行した命令である

(3) 通常ブロックの場合、命令コードと、次ブロック先頭アドレスにより、以下の命令群に分類し、図2に示すアルゴリズムにしたがって解析を進める。

CJMISS: **条件分岐命令で、分岐先と次ブロック先頭アドレスが違う**
CJMATCH: **条件分岐命令で、分岐先と次ブロック先頭アドレスが同じ**
UJMISS: **無条件分岐命令で、分岐先と次ブロック先頭アドレスが違う**
UJMATCH: **無条件分岐命令で、分岐先と次ブロック先頭アドレスが同じ**
FAIL: **逆アセンブルすべき命令がコード・ブロックになくなった**
NORMAL: **上記以外の命令**

図2において『確実』状態で逆アセンブルした命令はすべて実行された命令である。これに対して、『不確実I,II』状態で逆アセンブルした命令は、実際には実行していないものも含まれる可能性がある。ただし、『不確実I,II』状態において、プリフェッチ可能なバイト数以上の命令長が連続して"不確実"になった場合には、それは"確実"に実行された命令であると考えてよい。

(4) 次ブロック先頭アドレスから同一手順を繰り返す

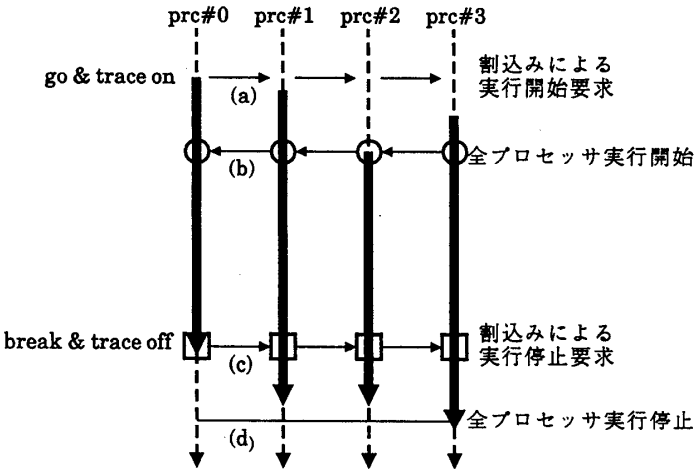

図3. マルチプロセッサ用のトレース方式

4. マルチプロセッサ用のトレース方式

マルチプロセッサ環境でプログラムの走行履歴を見る場合に、プロセッサ間で共有されている資源のある瞬間の状態や、システム全体での事象の前後関係を知りたい場合がある。

例えば、図3の(c)の時点での資源Xの状態が知りたいとする。この場合、(c)の場所にブレーク・ポイントを設定し、これにヒットしたときにデバッグ・モニタを介して全プロセッサへ停止要求を発行する。一般的には、割込みマスクや信号遅延が存在するため、一瞬のうちに全プロセッサが止まることはなく、必ず(c)〜(d)のような時間が存在する。したがって、全プロセッサにおいて(c)のタイミングをトレース・メモリ内に記録すれば、(d)での資源Xの状態と、トレース・メモリ内にある(c)から(d)までの資源Xへのアクセス状態から、(c)での資源Xの状態が推定可能である。また、counter情報を併用することにより、事象の前後関係も明らかになる。

上記のようにシステム・グローバルな事象の記憶はハードウェア・トレーサの option信号を用いて実現可能である。

5. おわりに

大容量トレーサにより、約2万命令、 6msec 以上の詳細なトレース結果が採取可能となった。また、今回述べた方法は完全な命令再構成方法ではないにも関わらず、実際のデバッグ時には、ほとんどの命令が誤差なく再構成できることを確認した。現在は、シングルプロセッサ用トレーサ・ソフトができたところであり、今後、マルチプロセッサ用トレーサ・ソフトを実装するとともに、関数レベルのトレースやトレーサを利用した計測機能などを拡張していきたい。

参考文献

[1]広垣,桃井,宮地, "マルチプロセッサ・リアルタイムUNIX MUSTARD とその開発環境", 並列処理シンポジウムJSPP'90, 1990.